

INSTITUTO POLITECNICO NACIONAL  
ESCUELA SUPERIOR DE COMPUTO

HERNANDEZ DELGADO RAÚL

Web Application Development

8CM4

06-02-2012

## INTRODUCCION

Un **applet Java** es un applet escrito en el lenguaje de programación Java. Los applets de Java pueden ejecutarse en un navegador web utilizando la Java Virtual Machine (JVM), o en el AppletViewer de Sun.

En Java, un *applet* es un programa que puede incrustarse en un documento HTML, es decir en una página web. Cuando un navegador carga una página web que contiene un *applet*, este se descarga en el navegador web y comienza a ejecutarse. Esto permite crear programas que cualquier usuario puede ejecutar con tan solo cargar la página web en su navegador.

### Java.policy

La política de seguridad del sistema se configura a partir de uno o más ficheros de configuración. Estos ficheros indican que permisos se conceden a que código y para que recursos específicos.

El fichero de configuración de la política de seguridad esencialmente contiene una lista de entradas. Puede contener una entrada *keystore* y una o más entradas *grant*.

El keystore es la base de datos de claves privadas y sus certificados asociados. La entrada keystore sirve para indicar dónde consultar las claves públicas de las entidades firmantes especificadas en las entradas grant del fichero. La entrada keystore debe aparecer en el fichero si cualquier entrada grant especifica el alias de una entidad que firma.

El fichero sólo puede contener una entrada keystore en el fichero (cualquier otra después de la primera será ignorada), y de ubicarse independientemente de cualquier otra entrada. Su sintaxis es:

```
keystore "url_fichero_keystore"
```

donde url\_fichero\_keystore especifica la localización URL de la base de datos. El URL es relativo a la localización del fichero de configuración, es decir, si el fichero de configuración se encuentra en el URL `http://case.iti.upv.es/seguridad/iti.policy` y la entrada del keystore es keystore ".keystore" el keystore se encontrará en el URL `http://case.iti.upv.es/seguridad/.keystore`. También puede especificarse un URL absoluto.

Cada entrada grant consiste básicamente en un la especificación del origen de una clase y sus permisos. Cada entrada grant del fichero sigue el siguiente formato, donde la palabra reservada grant indica el comienzo de una nueva entrada. Dentro de cada entrada, la palabra reservada permission marca el comienzo de un nuevo permiso en la entrada.

```
grant [SignedBy "lista_alias_entidades_firmantes"] [, CodeBase "URL"] {  
  permission clase_permiso ["recurso"][, "acción"][, SignedBy "lista_alias"];  
  permission ...  
  ...  
};
```

Se permiten espacios en blanco inmediatamente antes o después de las comas.

La clase\_permiso debe ser el nombre de una clase de permisos, por ejemplo `java.io.FilePermission` y no puede abreviarse escribiendo únicamente `FilePermission`.

El campo acción es opcional y puede omitirse si la clase del permiso no lo necesita. El campo CodeBase también es opcional si se omite significa *cualquier código*.

## DESARROLLO

La practica consiste en desarrollar un applet que abra un archivo de texto y lo muestre dentro de un textarea y guardar el contenido en otro archivo. Además agregar un reloj digital que marque la hora actual en cada instante.

En base a el código proporcionado en laboratorio se creara el applet considerando lo siguiente:

- El código original es para una aplicación de escritorio.
- Solo se proporciona el código para abrir y leer el archivo.
- Para tener acceso a los recursos de la plataforma es necesario proporcionar permisos.

Las principales modificaciones del código radican en que la clase hereda de Applet y no se utilizará la clase Frame para la implementación de la practica y la gregación del botón para guardar. Por lo tanto en vez de escribir el constructor de la clase se sobrescribirá el método init() para inicializar las variables. Para el constructor de la instancia de FileDialog se utiliza una clase anónima de Frame.

A continuación se muestra el código pertinente del las modificaciones ya antes mencionadas::

```
public void init(){
    p=new Panel(new GridLayout(1,3));
    l=new Label("Archivo:",2);
    b=new Button("Abrir");
    b.addActionListener(this);
    g=new Button("Guardar");
    g.addActionListener(this);
    ta=new TextArea();
    p.add(l);
    p.add(b);
    p.add(g);
    add("North",p);
    add("Center",ta);
}
```

```
if(s.equals("Abrir")){
    fd=new FileDialog(new Frame(),"Archivos",FileDialog.LOAD);
    fd.setVisible(true);
    arch=fd.getFile();
    ruta=fd.getDirectory();
    try{
        br=new BufferedReader(new FileReader(ruta+arch));
        while((texto=br.readLine())!=null){
            ta.append(texto+"\n");
        }
        br.close();
    }catch(IOException ioe){}
}
```

Para el desarrollo del reloj se implementará la interfaz Runnable y se hará una instancia de Thread para manejar el reloj. Los métodos a sobrescribir son run(), start(), paint() y stop().

El método run() cuenta con el siguiente código:

```
public void run() {
    while(Thread.currentThread()==reloj){//reloj es la instancia de Thread
        repaint();
        try{
            reloj.sleep(1000);
        }catch(InterruptedException ie){}
    }
}
```

El método start() cuenta con el siguiente código:

```
public void start(){
    if(reloj==null){
        reloj=new Thread(this,"Reloj");
        reloj.start();
    }
}
```

El método paint() cuenta con el siguiente código:

```
public void paint(Graphics g){
    Date ahora=new Date();
    g.drawString(""+ahora.getHours()+":"+ahora.getMinutes()+":"+ahora.getSeconds() , 5, 10);
}
```

El método stop() cuenta con el siguiente código:

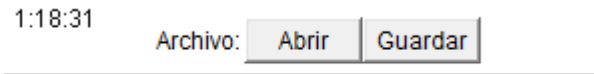
```
public void stop(){
    reloj=null;
}
```

Por último para agregar la funcionalidad de guardar la el contenido del TextArea se agregó el botón Guardar que encarga presisamente de esto. Para realizar esto se utilizaron las clase FileWriter y PrintWriter para crear el archivo: "Salida.txt" dentro del método actionPerformed() implentado por medio de la interfaz ActionListener y escribir sobre el como se muestra a continuación:

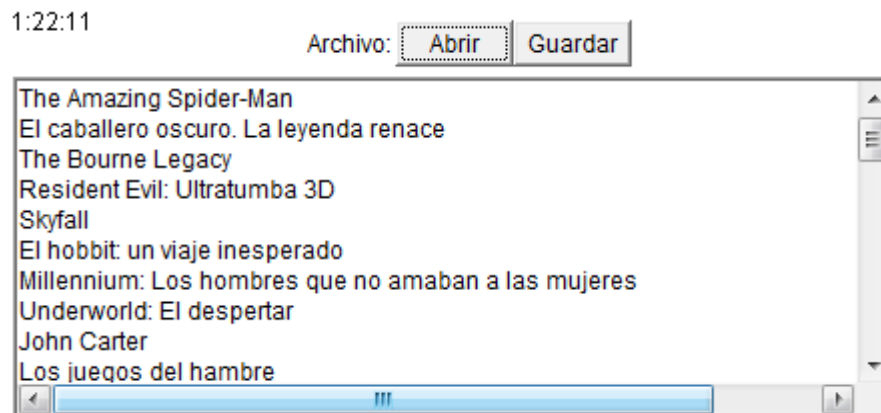
```
if(s.equals("Guardar")){
    FileWriter fw=null;
    PrintWriter pw=null;
    try{
        fw=new FileWriter("Salida.txt");
        pw=new PrintWriter(fw);
        String[] lineas=ta.getText().split("\n");
        int i=0;
        while(i<lineas.length){
            pw.println(lineas[i]);
            i++;
        }
    }
}
```

```
}catch(IOException ioe){}  
finally{  
    try{  
        fw.close();  
        pw.close();  
    }catch(IOException ioe){}  
}  
}
```

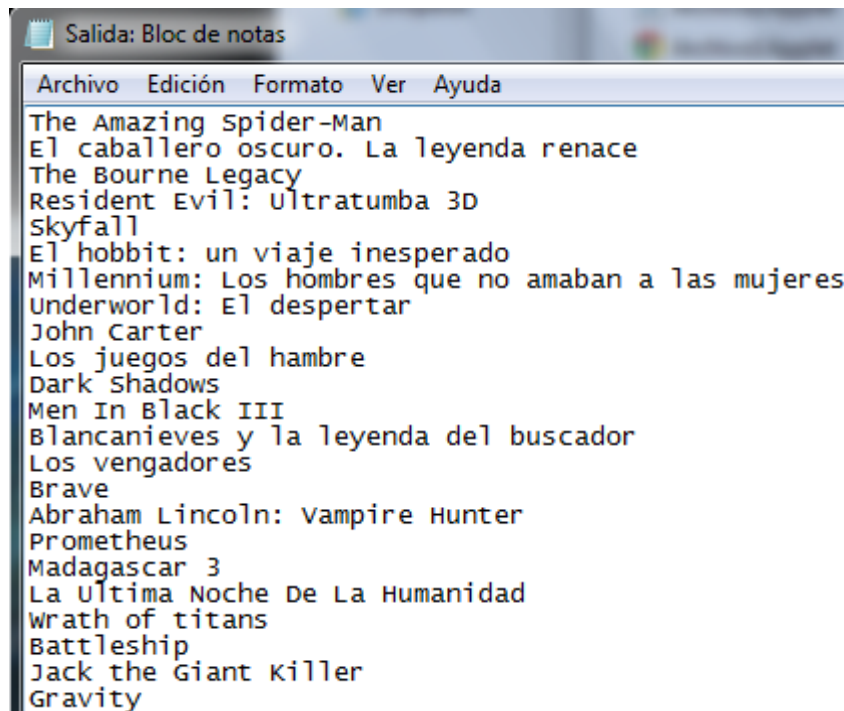
## Capturas de pantalla



*Ilustración 1: Applet Inicializado*



*Ilustración 2: Mostrando Contenido del archivo*



*Ilustración 3: Contenido del archivo Salida.txt*

Cabe señalar que se modificó el archivo java.policy para dar permiso al applet para abrir leer y escribir archivos como muestra la siguiente imagen:

```
grant codeBase "file:/C:/Users/Raul Hernandez/Desktop/wad/-" {  
    permission java.security.AllPermission;  
};
```

## CONCLUSIONES

Para poder realizar procesos de lectura y escritura dentro de la plataforma mediante un applet es necesario darle los permisos pertinentes ya que si se hace, simplemente no se realizaran dichos procesos.

Cabe señalar que también es posible firmar y autorizar las applets y permitir que los autores de los mismos pueden tener permisos usan como se explico en la introducción la herramienta keystore.

Para el reloj también hubiera sido útil utilizar la herencia de la clase Thread para generar el reloj aunque hubiera implicado posiblemente la utilización de una clase anidada que heredará de esta ya que como se sabe, en java no existe la herencia múltiple.